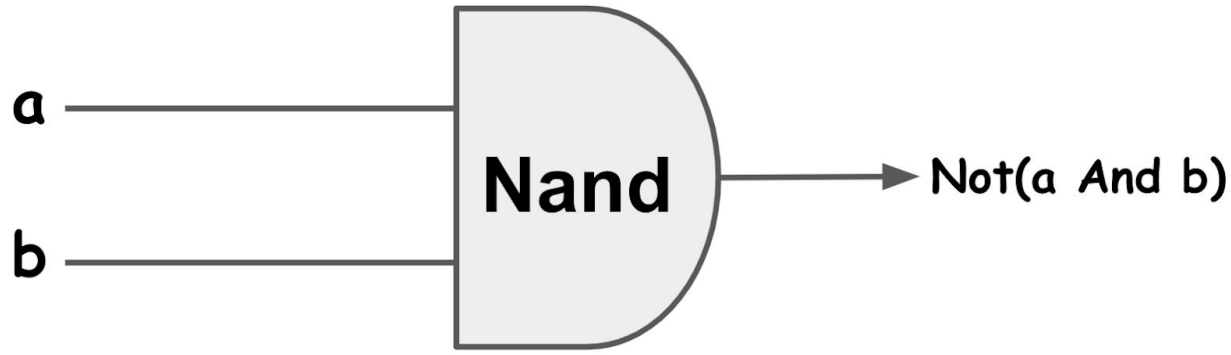
Project I: Boolean Logic

A typical computer architecture is based on a set of elementary logic gates like And, Or, Mux, etc., as well as their bit-wise versions And16, Or16, Mux16, etc. (assuming a 16-bit machine). This project engages you in the construction of a typical set of basic logic gates. These gates form the elementary building blocks from which more complex chips will be later constructed.

Objective

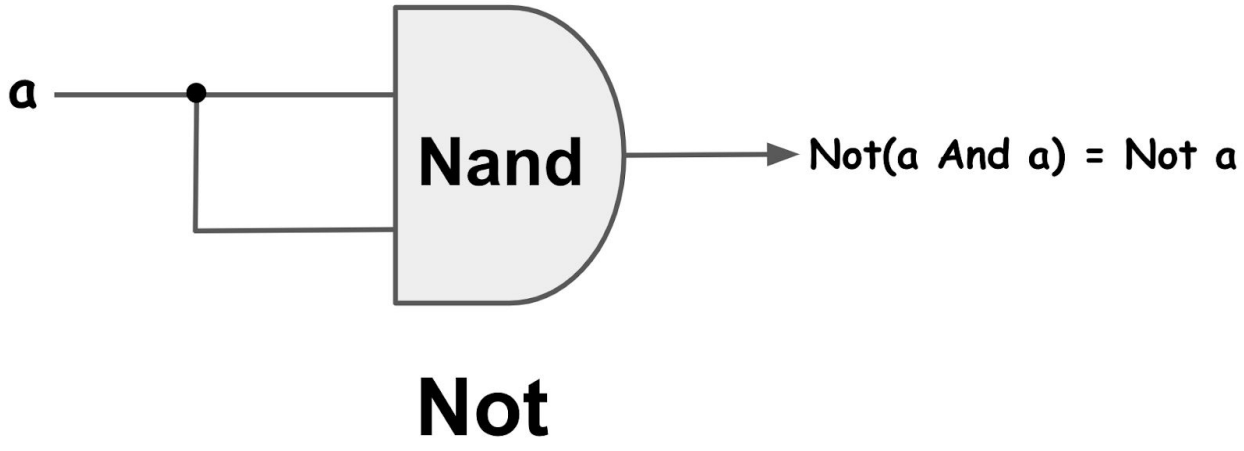
Build all the logic gates in the list below, yielding a basic chip-set. The only building blocks that you can use in this project are primitive Nand gates and the composite gates that you will gradually build on top of them.

Nand



truth table		
a	b	a Nand b
0	0	1
0	1	1
1	0	1
1	1	0

Not



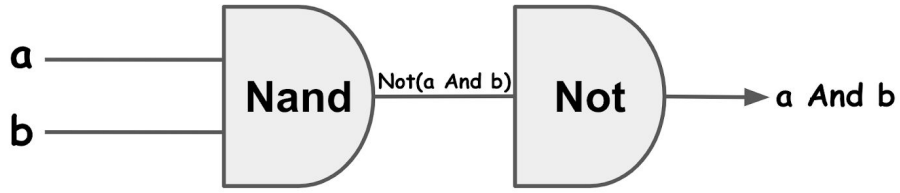
Not

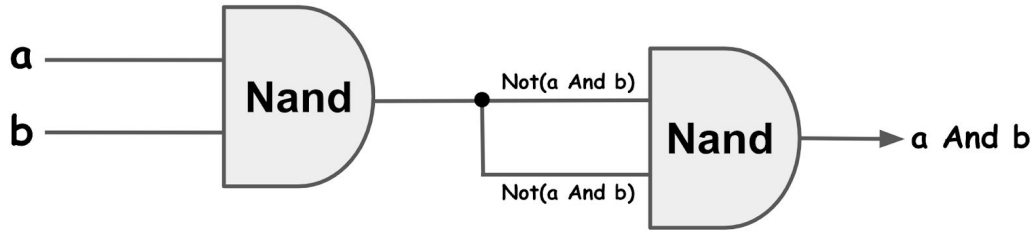
truth table	
a	out
0	1
1	0

Not.hdl — Edited

```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/01/Not.hdl

/**
 * Not gate:
 * out = not in
 */

CHIP Not {
    IN in;
    OUT out;

    PARTS:
    Nand(a=in,b=in,out=out);
}
```

And



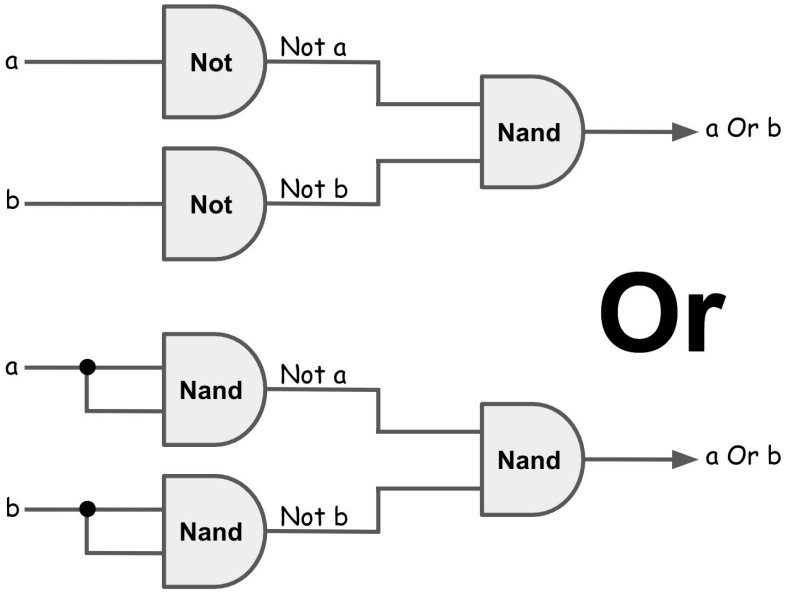


Truth Table		
a	b	a And b
0	0	0
0	1	0
1	0	0
1	1	1

And.hdl

```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/01/And.hdl

/**
 * And gate:
 * out = 1 if (a == 1 and b == 1)
 *       0 otherwise
 */

CHIP And {
    IN a, b;
    OUT out;

    PARTS:
    Nand(a=a,b=b,out=nandOut);
    Not(in=nandOut,out=out);
}
```

Or

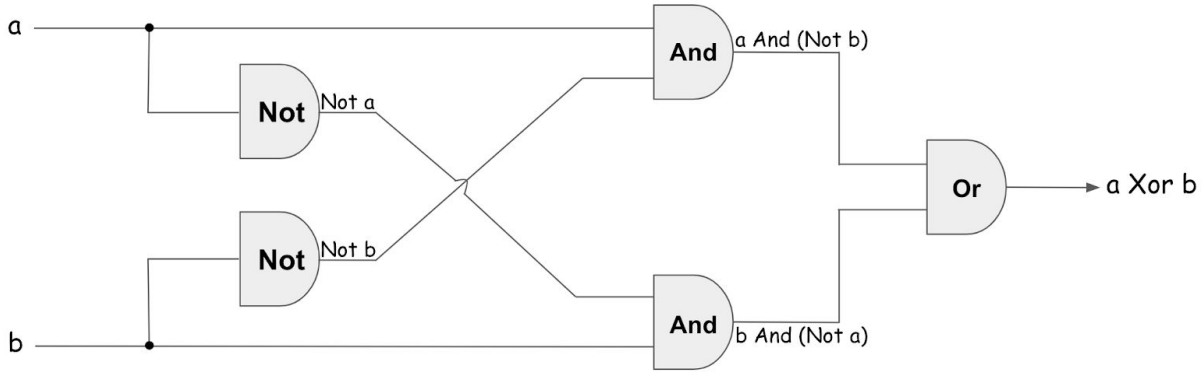


Truth Table		
a	b	a Or b
0	0	0
0	1	1
1	0	1
1	1	1

Or.hdl

```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/01/Or.hdl

 /**
 * Or gate:
 * out = 1 if (a == 1 or b == 1)
 *       0 otherwise
 */

CHIP Or {
    IN a, b;
    OUT out;

    PARTS:
    Not(in=a,out=nota);
    Not(in=b,out=notb);
    Nand(a=nota,b=notb,out=out);
}
```

Xor

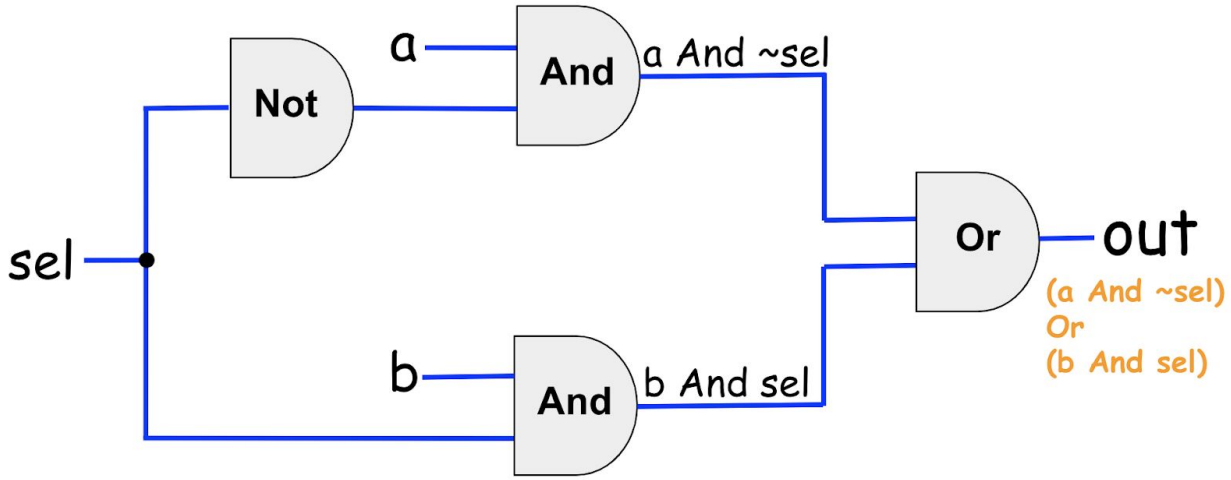


Truth Table		
a	b	a Xor b
0	0	1
0	1	0
1	0	0
1	1	1

Xor.hdl

```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/01/Xor.hdl

/**
 * Exclusive-or gate:
 * out = not (a == b)
 */

CHIP Xor {
    IN a, b;
    OUT out;

    PARTS:
    Not(in=a,out=nota);
    Not(in=b,out=notb);
    And(a=a,b=notb,out=aAndNotb);
    And(a=nota,b=b,out=notaAndb);
    Or (a=aAndNotb,b=notaAndb,out=out);
}
```

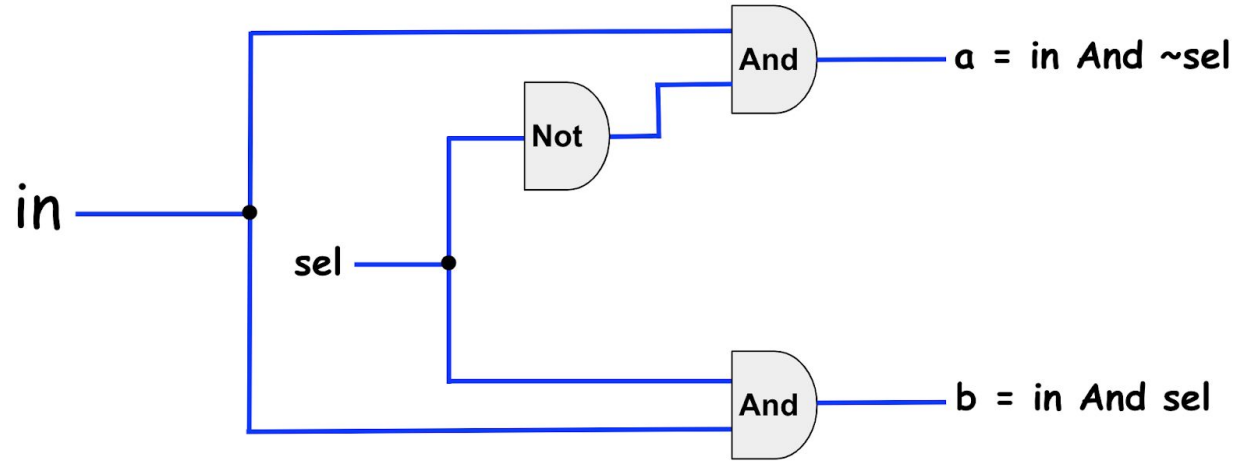
Mux



Truth Table			
a	b	s	out
0	0	1	0
0	0	0	0
0	1	1	0
0	1	0	1
1	0	1	1
1	0	0	0
1	1	1	1
1	1	0	1

```
CHIP Mux {
    IN a, b, sel;
    OUT out;

    PARTS:
    Nand(a=sel, b=sel, out=notSel);
    And(a=a, b=notSel, out=aAndnotSel);
    And(a=b, b=sel, out=bAndsel);
    Or(a=aAndnotSel, b=bAndsel, out=out);
}
```

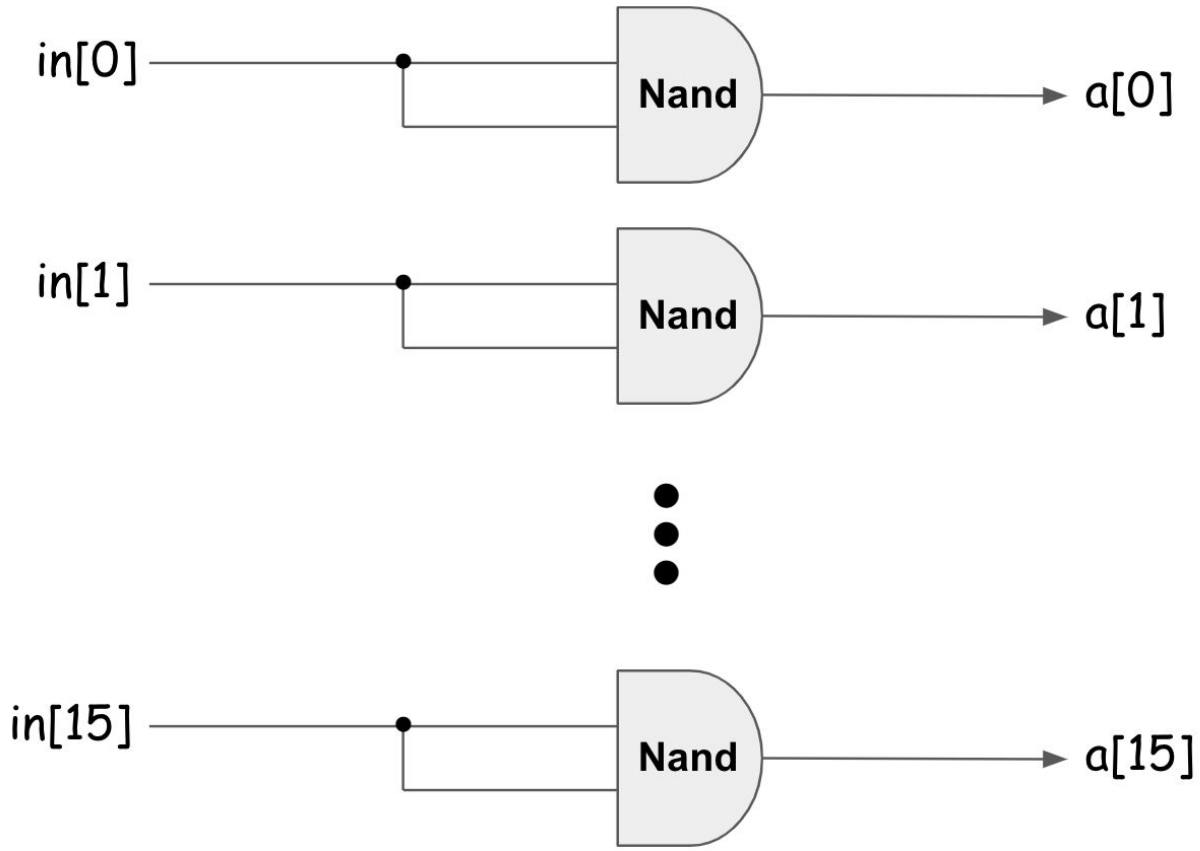
DMux



Truth Table			
in	sel	a	b
0	0	0	0
0	1	0	0
1	0	1	0

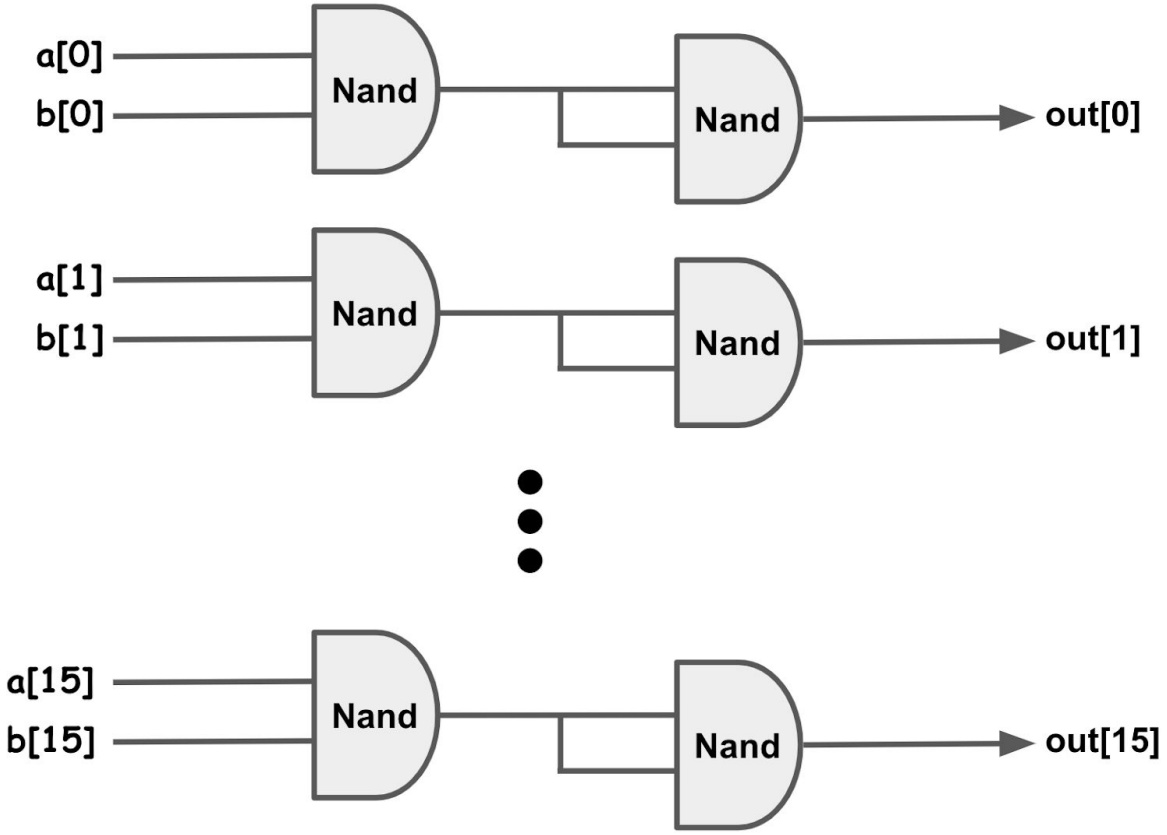
```
CHIP DMux {
    IN in, sel;
    OUT a, b;

    PARTS:
    Nand(a=sel, b=sel, out=notSel);
    And(a=in, b=notSel, out=a);
    And(a=in, b=sel, out=b);
}
```

Not16



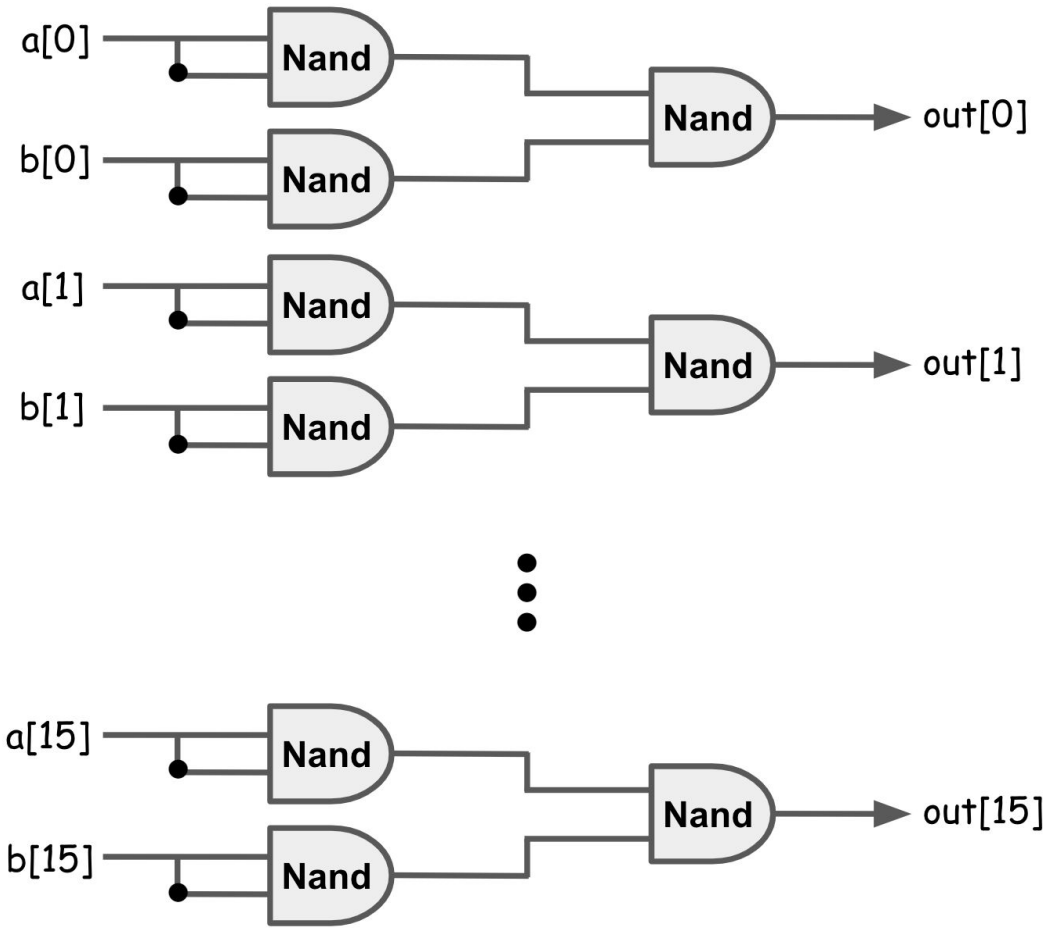

```
CHIP Not16 {
    IN in[16];
    OUT out[16];

    PARTS:
    Nand(a=in[0],b=in[0],out=out[0]);
    Nand(a=in[1],b=in[1],out=out[1]);
    Nand(a=in[2],b=in[2],out=out[2]);
    Nand(a=in[3],b=in[3],out=out[3]);
    Nand(a=in[4],b=in[4],out=out[4]);
    Nand(a=in[5],b=in[5],out=out[5]);
    Nand(a=in[6],b=in[6],out=out[6]);
    Nand(a=in[7],b=in[7],out=out[7]);
    Nand(a=in[8],b=in[8],out=out[8]);
    Nand(a=in[9],b=in[9],out=out[9]);
    Nand(a=in[10],b=in[10],out=out[10]);
    Nand(a=in[11],b=in[11],out=out[11]);
    Nand(a=in[12],b=in[12],out=out[12]);
    Nand(a=in[13],b=in[13],out=out[13]);
    Nand(a=in[14],b=in[14],out=out[14]);
    Nand(a=in[15],b=in[15],out=out[15]);
}
```

And16



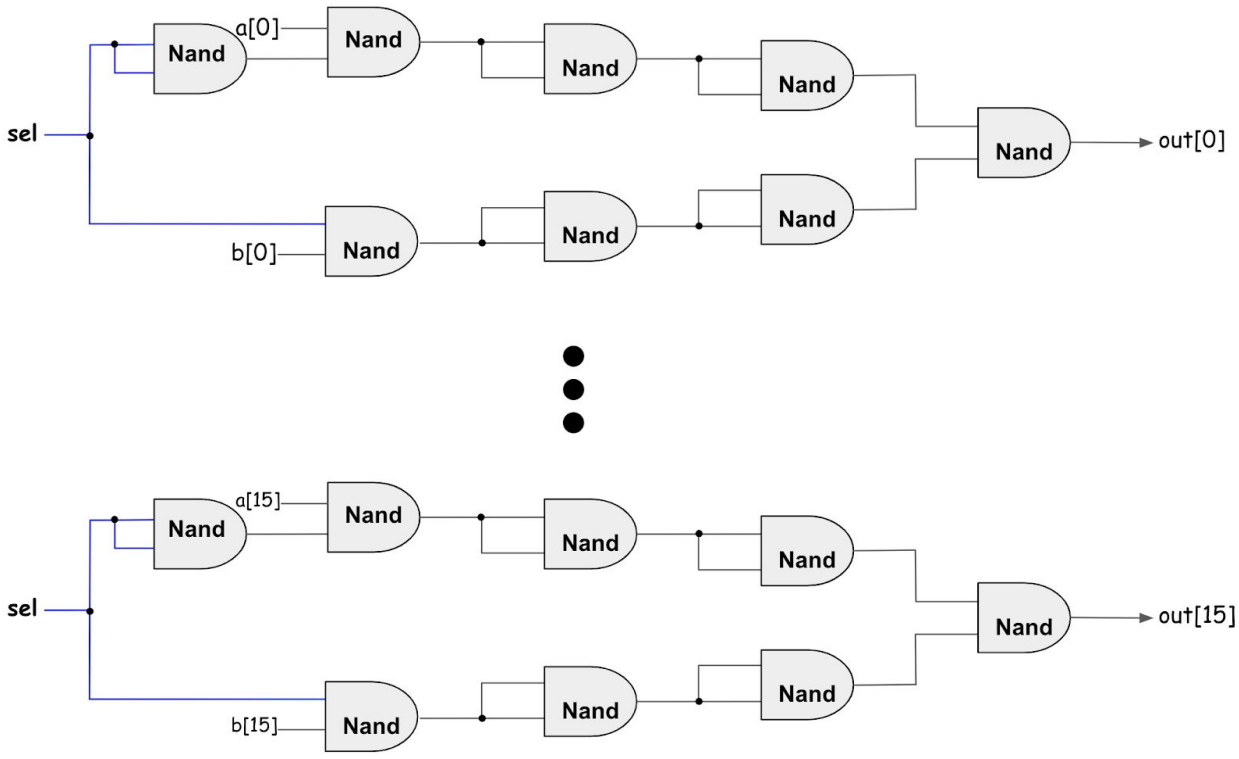

```
CHIP And16 {
    IN a[16], b[16];
    OUT out[16];

    PARTS:
    And(a=a[0], b=b[0], out=out[0]);
    And(a=a[1], b=b[1], out=out[1]);
    And(a=a[2], b=b[2], out=out[2]);
    And(a=a[3], b=b[3], out=out[3]);
    And(a=a[4], b=b[4], out=out[4]);
    And(a=a[5], b=b[5], out=out[5]);
    And(a=a[6], b=b[6], out=out[6]);
    And(a=a[7], b=b[7], out=out[7]);
    And(a=a[8], b=b[8], out=out[8]);
    And(a=a[9], b=b[9], out=out[9]);
    And(a=a[10], b=b[10], out=out[10]);
    And(a=a[11], b=b[11], out=out[11]);
    And(a=a[12], b=b[12], out=out[12]);
    And(a=a[13], b=b[13], out=out[13]);
    And(a=a[14], b=b[14], out=out[14]);
    And(a=a[15], b=b[15], out=out[15]);
}
```

Or16



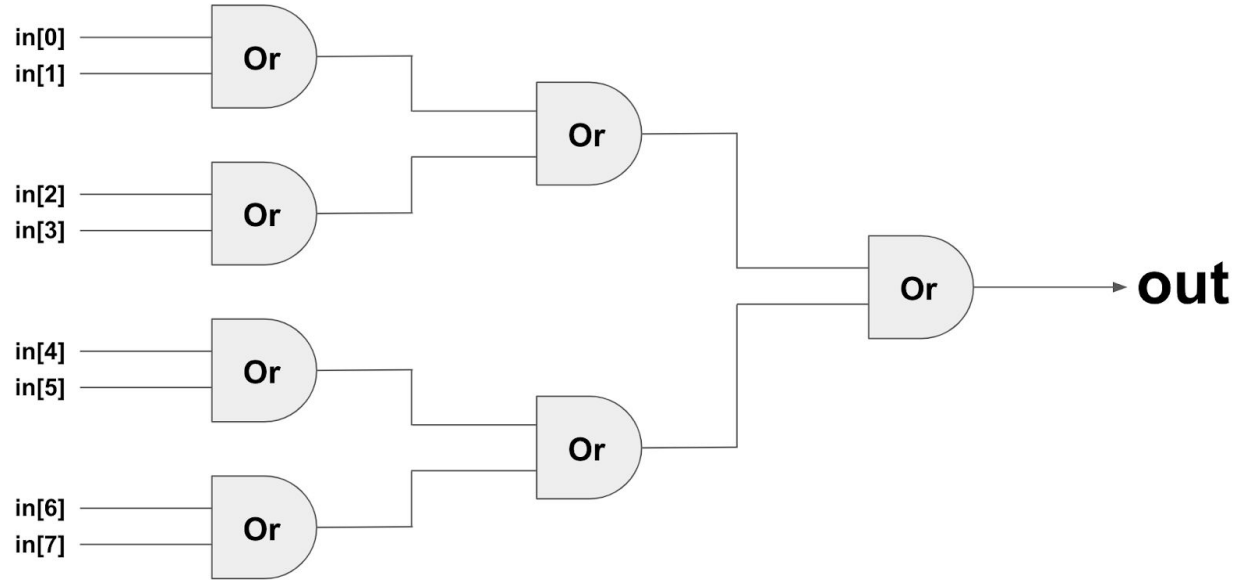

```
CHIP Or16 {
    IN a[16], b[16];
    OUT out[16];

    PARTS:
    Or(a=a[0], b=b[0], out=out[0]);
    Or(a=a[1], b=b[1], out=out[1]);
    Or(a=a[2], b=b[2], out=out[2]);
    Or(a=a[3], b=b[3], out=out[3]);
    Or(a=a[4], b=b[4], out=out[4]);
    Or(a=a[5], b=b[5], out=out[5]);
    Or(a=a[6], b=b[6], out=out[6]);
    Or(a=a[7], b=b[7], out=out[7]);
    Or(a=a[8], b=b[8], out=out[8]);
    Or(a=a[9], b=b[9], out=out[9]);
    Or(a=a[10], b=b[10], out=out[10]);
    Or(a=a[11], b=b[11], out=out[11]);
    Or(a=a[12], b=b[12], out=out[12]);
    Or(a=a[13], b=b[13], out=out[13]);
    Or(a=a[14], b=b[14], out=out[14]);
    Or(a=a[15], b=b[15], out=out[15]);
}
```

Mux16



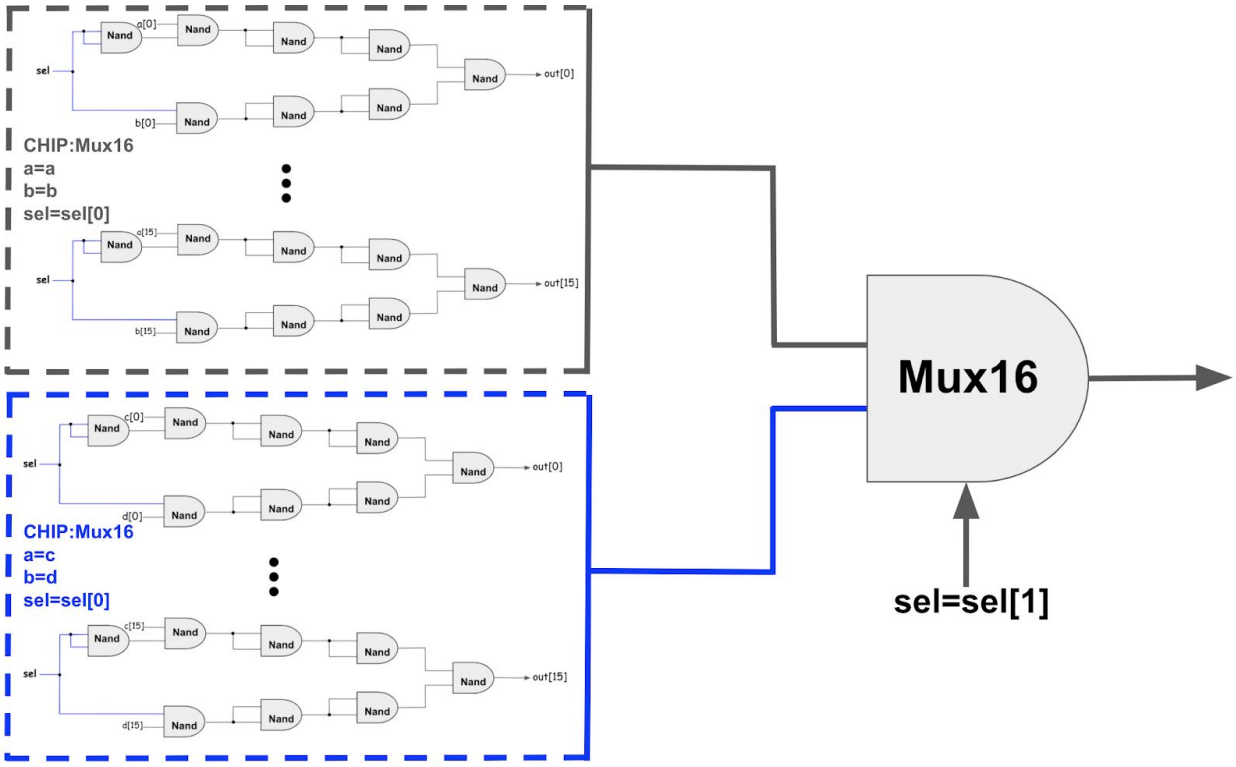

```
CHIP Mux16 {
    IN a[16], b[16], sel;
    OUT out[16];

    PARTS:
    Mux(a=a[0], b=b[0], sel=sel, out=out[0]);
    Mux(a=a[1], b=b[1], sel=sel, out=out[1]);
    Mux(a=a[2], b=b[2], sel=sel, out=out[2]);
    Mux(a=a[3], b=b[3], sel=sel, out=out[3]);
    Mux(a=a[4], b=b[4], sel=sel, out=out[4]);
    Mux(a=a[5], b=b[5], sel=sel, out=out[5]);
    Mux(a=a[6], b=b[6], sel=sel, out=out[6]);
    Mux(a=a[7], b=b[7], sel=sel, out=out[7]);
    Mux(a=a[8], b=b[8], sel=sel, out=out[8]);
    Mux(a=a[9], b=b[9], sel=sel, out=out[9]);
    Mux(a=a[10], b=b[10], sel=sel, out=out[10]);
    Mux(a=a[11], b=b[11], sel=sel, out=out[11]);
    Mux(a=a[12], b=b[12], sel=sel, out=out[12]);
    Mux(a=a[13], b=b[13], sel=sel, out=out[13]);
    Mux(a=a[14], b=b[14], sel=sel, out=out[14]);
    Mux(a=a[15], b=b[15], sel=sel, out=out[15]);
}
```

Or8Way



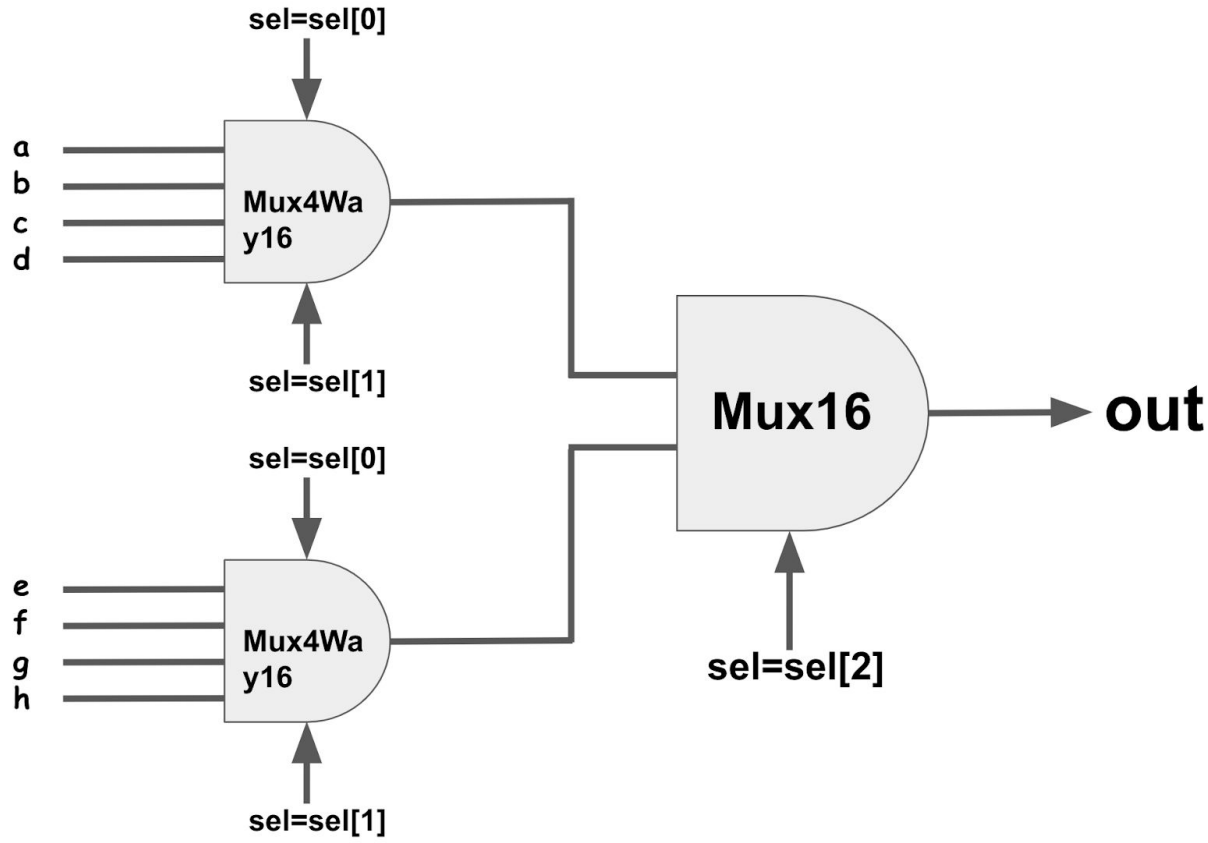

```
CHIP Or8Way {
    IN in[8];
    OUT out;

    PARTS:
    Or(a=in[0], b=in[1],out=c);
    Or(a=in[2], b=in[3],out=d);
    Or(a=in[4], b=in[5],out=e);
    Or(a=in[6], b=in[7],out=f);

    Or(a=c,b=d,out=g);
    Or(a=e,b=f,out=h);

    Or(a=g,b=h,out=out);
}
```

Mux4Way16



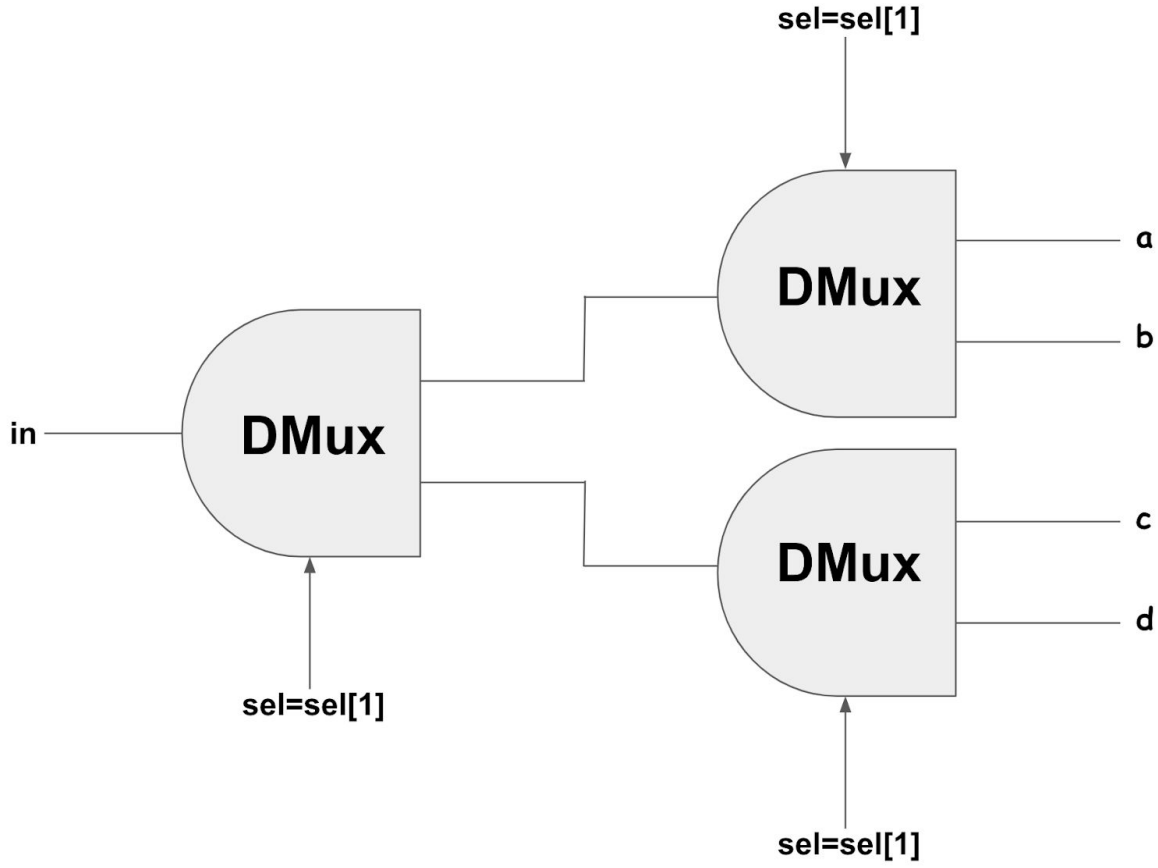

```
CHIP Mux4Way16 {
    IN a[16], b[16], c[16], d[16], sel[2];
    OUT out[16];

    PARTS:
    Mux16(a=a,b=b,sel=sel[0],out=chab);
    Mux16(a=c,b=d,sel=sel[0],out=chcd);
    Mux16(a=chab,b=chcd,sel=sel[1],out=out);

}
```

Mux8Way16



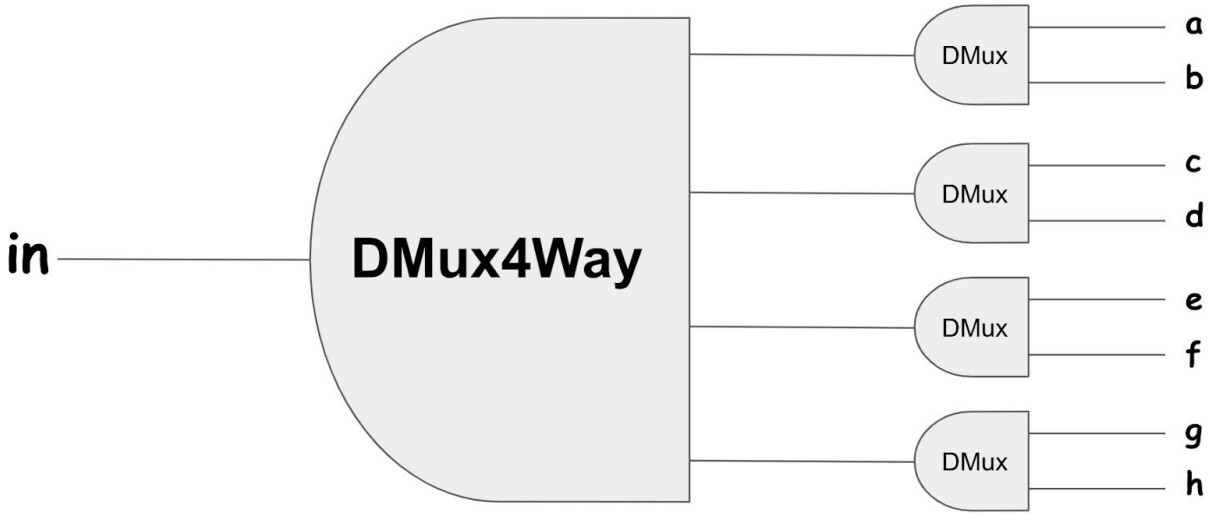

```
CHIP Mux8Way16 {
    IN a[16], b[16], c[16], d[16],
       e[16], f[16], g[16], h[16],
       sel[3];
    OUT out[16];

    PARTS:
    Mux4Way16(a=a,b=b,c=c,d=d,sel[0]=sel[0],sel[1]=sel[1],out=chabcd);
    Mux4Way16(a=e,b=f,c=g,d=h,sel[0]=sel[0],sel[1]=sel[1],out=chefgh);
    Mux16(a=chabcd,b=chefgh,sel=sel[2],out=out);
}
```

DMux4Way




```
CHIP DMux4Way {
    IN in, sel[2];
    OUT a, b, c, d;

    PARTS:
    DMux(in=in, sel=sel[1], a=one, b=two);
    DMux(in=one, sel=sel[0], a=a, b=b);
    DMux(in=two, sel=sel[0], a=c, b=d);
}
```

DMux8Way




```
CHIP DMux8Way {
    IN in, sel[3];
    OUT a, b, c, d, e, f, g, h;

    PARTS:
    DMux4Way(in=in,sel[0]=sel[1],sel[1]=sel[2],a=one,b=two,c=three,d=four);
    DMux(in=one,sel=sel[0],a=a,b=b);
    DMux(in=two,sel=sel[0],a=c,b=d);
    DMux(in=three,sel=sel[0],a=e,b=f);
    DMux(in=four,sel=sel[0],a=g,b=h);
}
```

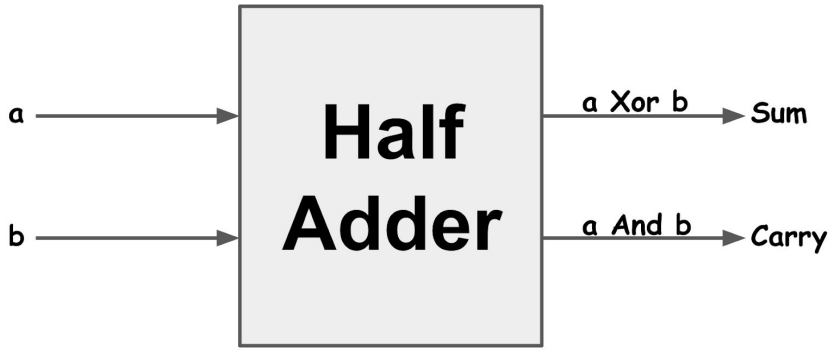
Project II: Boolean Arithmetic

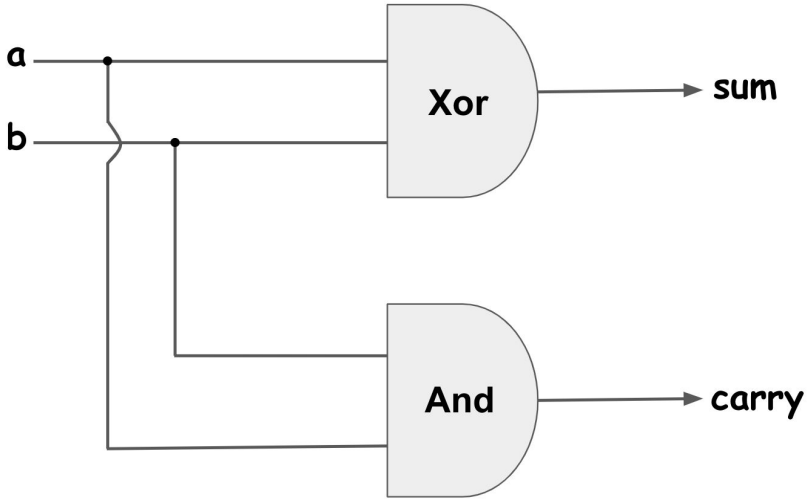
The centerpiece of the computer's architecture is the CPU, or Central Processing Unit, and the centerpiece of the CPU is the ALU, or Arithmetic-Logic Unit. In this project you will gradually build a set of chips, culminating in the construction of the ALU chip of the Hack computer. All the chips built in this project are standard, except for the ALU itself, which differs from one computer architecture to another.

Objective

Build all the chips described in the list below, leading up to an Arithmetic Logic Unit - the Hack computer's ALU. The only building blocks that you can use are the chips described in chapter 1 and the chips that you will gradually build in this project.

Half Adder

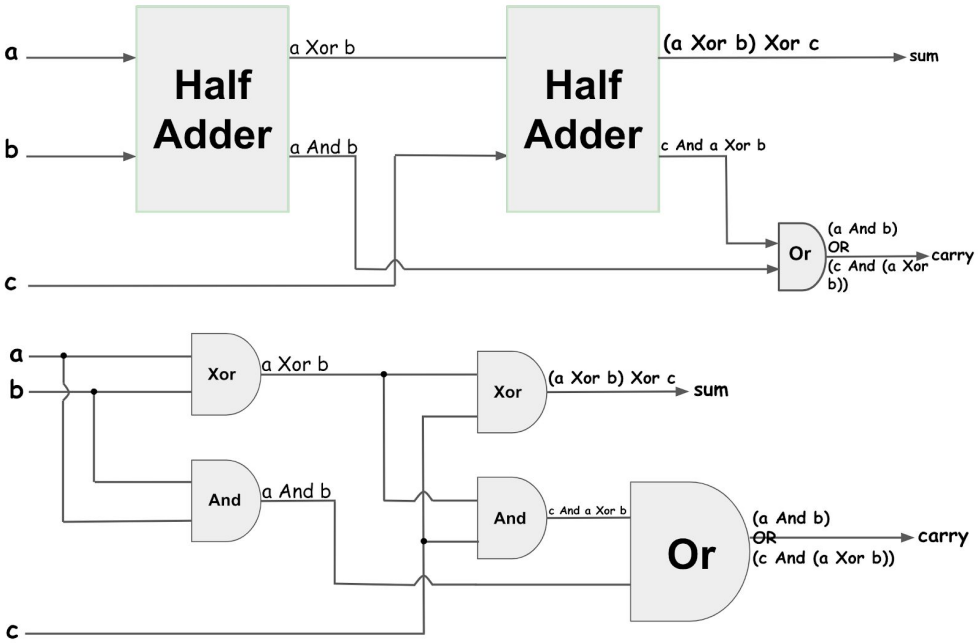




a	b	sum	carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

```
CHIP HalfAdder {
    IN a, b;    // 1-bit inputs
    OUT sum,    // Right bit of a + b
        carry;  // Left bit of a + b

    PARTS:
    Xor(a=a,b=b,out=sum);
    And(a=a,b=b,out=carry);
}
```

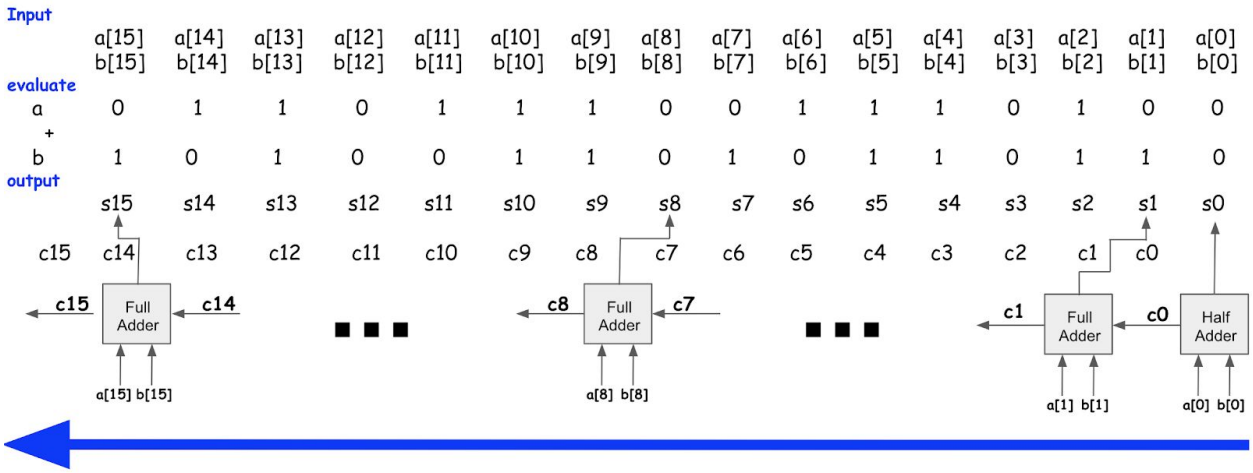
Full Adder



a	b	c	sum	carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

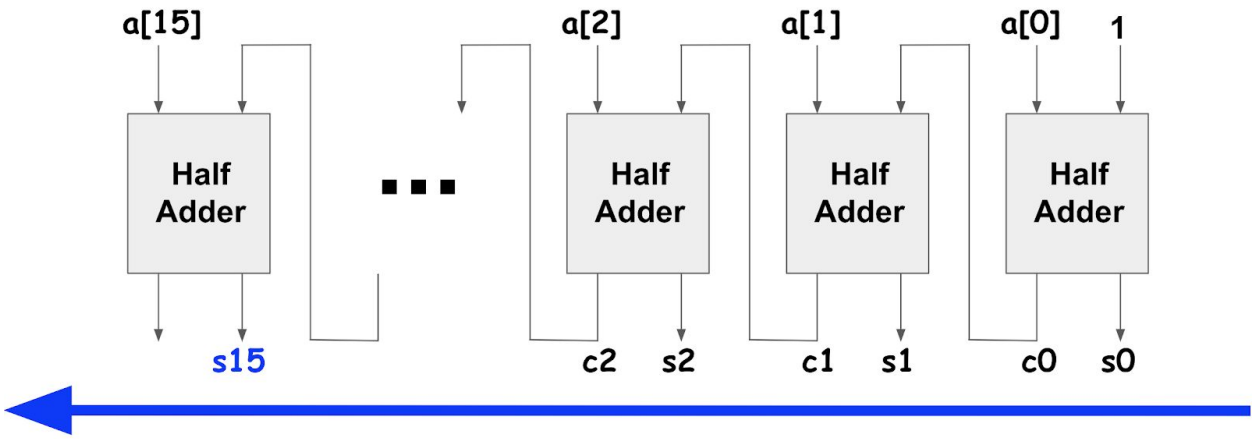
```
CHIP FullAdder {
    IN a, b, c;  // 1-bit inputs
    OUT sum,     // Right bit of a + b + c
        carry;   // Left bit of a + b + c

    PARTS:
    Xor(a=a, b=b, out=d);
    And(a=a, b=b, out=e);
    Xor(a=d, b=c, out=sum);
    And(a=d, b=c, out=f);
    Or (a=f, b=e, out=carry);
}
```

Add16



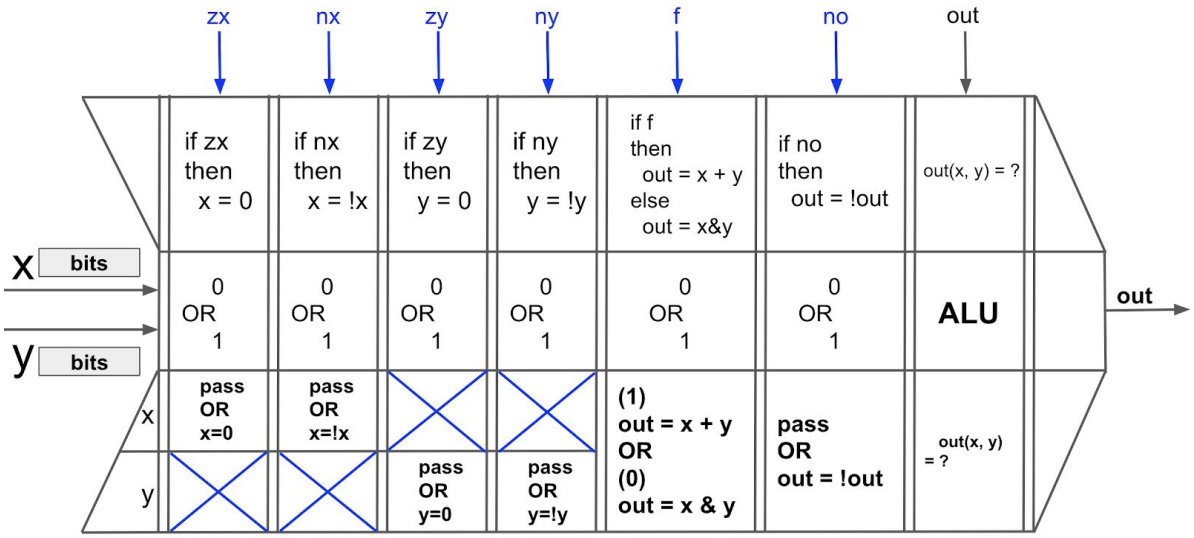

```
CHIP Add16 {
    IN a[16], b[16];
    OUT out[16];

    PARTS:
    FullAdder(a=a[0], b=b[0], sum=out[0],carry=c1);
    FullAdder(a=a[1], b=b[1], c=c1, sum=out[1], carry=c2);
    FullAdder(a=a[2], b=b[2], c=c2, sum=out[2], carry=c3);
    FullAdder(a=a[3], b=b[3], c=c3, sum=out[3], carry=c4);
    FullAdder(a=a[4], b=b[4], c=c4, sum=out[4], carry=c5);
    FullAdder(a=a[5], b=b[5], c=c5, sum=out[5], carry=c6);
    FullAdder(a=a[6], b=b[6], c=c6, sum=out[6], carry=c7);
    FullAdder(a=a[7], b=b[7], c=c7, sum=out[7], carry=c8);
    FullAdder(a=a[8], b=b[8], c=c8, sum=out[8], carry=c9);
    FullAdder(a=a[9], b=b[9], c=c9, sum=out[9], carry=c10);
    FullAdder(a=a[10], b=b[10], c=c10, sum=out[10], carry=c11);
    FullAdder(a=a[11], b=b[11], c=c11, sum=out[11], carry=c12);
    FullAdder(a=a[12], b=b[12], c=c12, sum=out[12], carry=c13);
    FullAdder(a=a[13], b=b[13], c=c13, sum=out[13], carry=c14);
    FullAdder(a=a[14], b=b[14], c=c14, sum=out[14], carry=c15);
    FullAdder(a=a[15], b=b[15], c=c15, sum=out[15]);
}
```

Inc16




```
CHIP Inc16 {
    IN in[16];
    OUT out[16];

    PARTS:
    HalfAdder(a=in[0], b=true, sum=out[0],carry=c1);
    HalfAdder(a=in[1], b=c1, sum=out[1], carry=c2);
    HalfAdder(a=in[2], b=c2, sum=out[2], carry=c3);
    HalfAdder(a=in[3], b=c3, sum=out[3], carry=c4);
    HalfAdder(a=in[4], b=c4, sum=out[4], carry=c5);
    HalfAdder(a=in[5], b=c5, sum=out[5], carry=c6);
    HalfAdder(a=in[6], b=c6, sum=out[6], carry=c7);
    HalfAdder(a=in[7], b=c7, sum=out[7], carry=c8);
    HalfAdder(a=in[8], b=c8, sum=out[8], carry=c9);
    HalfAdder(a=in[9], b=c9, sum=out[9], carry=c10);
    HalfAdder(a=in[10], b=c10, sum=out[10], carry=c11);
    HalfAdder(a=in[11], b=c11, sum=out[11], carry=c12);
    HalfAdder(a=in[12], b=c12, sum=out[12], carry=c13);
    HalfAdder(a=in[13], b=c13, sum=out[13], carry=c14);
    HalfAdder(a=in[14], b=c14, sum=out[14], carry=c15);
    HalfAdder(a=in[15], b=c15, sum=out[15]);
}
```

ALU




```
CHIP ALU {
    IN
        x[16], y[16],  // 16-bit inputs
        zx, // zero the x input?
        nx, // negate the x input?
        zy, // zero the y input?
        ny, // negate the y input?
        f,  // compute out = x + y (if 1) or x & y (if 0)
        no; // negate the out output?

    OUT
        out[16], // 16-bit output
        zr, // 1 if (out == 0), 0 otherwise
        ng; // 1 if (out < 0),  0 otherwise

    PARTS:
    // if zx then x = 0
    Mux16(a = x, b[0..15] = false, sel = zx, out = ozx);

    // if nx then x = !x
    Not16(in = ozx, out = nnx);
    Mux16(a = ozx, b = nnx, sel = nx, out = exx);

    // if zy then y = 0
    Mux16(a = y, b[0..15] = false, sel = zy, out = ozy);

    // if ny then y = !y
    Not16(in = ozy, out = nny);
    Mux16(a = ozy, b = nny, sel = ny, out = exy);

    // if f then out = x + y
    //      else out = x & y
    Add16(a = exx, b = exy, out = xplusy);
    And16(a = exx, b = exy, out = xandy);
    Mux16(a = xandy, b = xplusy, sel = f, out = fxy);

    // if no then out = !out
    Not16(in = fxy, out = nfxy);
    Mux16(a = fxy, b = nfxy, sel = no, out[0..7] = ret0, out[8..14] = ret1, out[15] = retsign, out = out);

    Or8Way(in[0..7] = ret0, out = ret0is0);
    Or8Way(in[0..6] = ret1, in[7] = retsign, out = ret1is0);

    Or(a = ret0is0, b = ret1is0, out = yzr);
    Not(in = yzr, out = zr);

    // if out < 0 then ng = 1 else ng = 0
    And(a = retsign, b = true, out = ng);
}
```